

Can we Approximate the Test of Integrated Circuits?

Imran Wali, Marcello Traiola, Arnaud Virazel, Patrick Girard, Mario Barbareschi, Alberto Bosio

► **To cite this version:**

Imran Wali, Marcello Traiola, Arnaud Virazel, Patrick Girard, Mario Barbareschi, et al.. Can we Approximate the Test of Integrated Circuits?. WAPCO: Workshop On Approximate Computing, Jan 2017, Stockholm, Sweden. lirmm-02004418

HAL Id: lirmm-02004418

<https://hal-lirmm.ccsd.cnrs.fr/lirmm-02004418>

Submitted on 1 Feb 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Can we Approximate the Test of Integrated Circuits?

Imran Wali¹, Marcello Traiola¹, Arnaud Virazel¹, Patrick Girard¹, Mario Barbareschi², Alberto Bosio¹

¹LIRMM – University of Montpellier / CNRS – France

<name.lastname>@lirmm.fr

²Università Federico II – Naples

<name.lastname>@unina.it

Abstract—In the recent years *Approximate Computing* (AC) has emerged as new paradigm for energy efficient IC design. It addresses the problem of maintaining reliability and thus coping with run-time errors exploiting an acceptable amount of overheads in terms of area, performances and energy consumption. This work starts from the consideration that AC-based systems can intrinsically accept the presence of faulty hardware (i.e., hardware that can produce errors). This paradigm is also called “*computing on unreliable hardware*”. The hardware-induced errors have to be analyzed to determine their propagation through the system layers and eventually determining their impact on the final application. In other words, an AC-based system does not need to be built using defect-free ICs. Under this assumption, we can relax test and reliability constraints of the manufactured ICs. One of the ways to achieve this goal is to test only for a subset of faults instead of targeting all possible faults. In this way, we can reduce the manufacturing cost since we eventually reduce the number test patterns and thus the test time. We call this approach *Approximate Test*. The main advantage is the fact that we do not need a prior knowledge of the workload (i.e., we are application independent). Therefore, the proposed approach can be applied to any kind of ICs, reducing the test time and increasing the yield. We present preliminary results on some simple case studies. The main goal is to show that by letting some faults undetected we can save test time without having a huge impact on the application quality.

Keywords—*fault coverage; test pattern; approximate test; test generation; test complexity*

I. INTRODUCTION

Today’s Integrated Circuits (ICs) are starting to reach the physical limits of CMOS technology. Among the multiple challenges arising from technology nodes lower the 20 nm, we can highlight the high leakage current (i.e., high static power consumption), reduced performance gain, reduced reliability, complex manufacturing process leading to low yield and complex testing process, and extremely costly masks [1]. In other words, ICs manufactured with the latest technology nodes are the less and less efficient (w.r.t. both performances and energy consumption) than forecasted by the Moore’s law. Moreover, the manufactured devices are becoming less and less reliable, meaning that errors can appear during the normal lifetime of a device with a higher probability than in previous technology nodes [2]. Fault tolerant mechanisms are therefore required to ensure the correct behavior of such device at the cost of extra area, power and timing overheads. Finally, process variations force the engineers to add extra guard bands (e.g., higher supply voltage or lower clock frequency than required under normal circumstances) to guarantee the correct functioning of manufactured devices.

In the recent years, the *Approximate Computing* (AC) paradigm has been emerged [2][3][4]. It addresses the problem of maintaining reliability and thus coping with run-time errors, exploiting an acceptable amount of overheads in terms of area, performances and energy consumption. AC is based on the intuitive observation that, while performing exact computation requires a high amount of resources, allowing selective approximation or occasional violation of the specification can provide gains in efficiency (i.e., less power consumption, less area, higher manufacturing yield) without significantly affecting the output quality [2][3][4].

This work starts from the consideration that AC-based systems can intrinsically accept the presence of faulty hardware (i.e., hardware that can produce errors) [5]. The hardware-induced errors have to be analyzed to determine their propagation through the system layers and eventually determining their impact on the final application. In other words, an AC-based system does not need to be built using defect-free ICs. In deed, AC-based systems can manage at higher-level the errors due to defectives ICs, or those errors simply do not significantly impact on the final applications. Under this assumption, we can relax test and reliability constraints of the manufactured ICs. One of the ways to achieve this goal is to test only for a subset of faults instead of targeting all possible faults. In this way, we can reduce the manufacturing cost since we eventually reduce the number of test pattern and thus the test time. In the literature, some interesting works have already been published so far, targeting the test generation for a subset of faults [6]-[7]. The main idea behind these works is the classification of faults as benign and malignant to be further exploited during the test generation. More in detail, the authors propose to generate all possible faults and then classifying them into two classes: 1) *benign faults* - those that cause no error or an acceptable amount of error, and 2) *malignant faults* - those faults that cause a significant deviation from acceptable behavior. The metric used for classifying faults into these two classes is the error magnitude [2], which is the difference between actual value (affected by the fault) and the golden value (fault-free).

The contribution of this work is to investigate an approach opposite w.r.t. the state-of-the-art. Instead to classify the faults according to the applied workload, we exploit a functional and a structural analysis to determine the most vulnerable circuit elements and thus generate the test patterns for these elements. We call this approach *Approximate Test* (AT). The main advantage is the fact that we do not need a prior knowledge of the workload (i.e., we are application independent). Therefore, the proposed approach can be applied to any kind of ICs reducing the test time and increasing the yield. In this paper,

we present preliminary results on some simple case studies. The results aim at looking at the impact of the AT on the test length and the fault coverage. Moreover, we also show the impact on the final application by using the well-known *error probability* (P_e) and the *error magnitude* (ϵ) metrics [9]. The main goal is to show that by letting some faults undetected we can save test time without having a huge impact on the application quality.

The paper is organized as follows. Section II presents the flow of the proposed approach and gives details of each step. Experimental results are discussed in Section III. Finally, conclusions are given in Section IV.

II. THE PROPOSED APPROACH

The main idea of the proposed AT approach is to let some faults untested in order to speed up the test and to increase the overall yield. As already discussed, the risk of exploiting defective ICs to build AC-based systems is mitigated by the fact the AC can intrinsically accept the presence of errors. However, the important point is that the impact of the untested faults on the final applications has to be in the acceptable regions (i.e., the final output quality is still acceptable by the user). The real challenge is therefore determining what are the faults that must be tested and what are those that can be ignored during the test application.

The straightforward approach for determining the targeted faults is act accordingly to the functionality of the circuit. The classical example is an arithmetic circuit where it is better to guarantee that the most significant outputs are correct to narrow down the ϵ [2]. Unfortunately this approach cannot be adopted for all the kind of integrated circuits since it is not always easy to determine the most significant outputs. In deed the latter are clearly strictly related to the workload. To overcome the above issue, we propose to determine the targeted faults by using a structural analysis. In this way we want to be independent w.r.t. the circuit function. To validate our proposal, we developed a flow for applying both the approaches (i.e., functional and structural analysis).

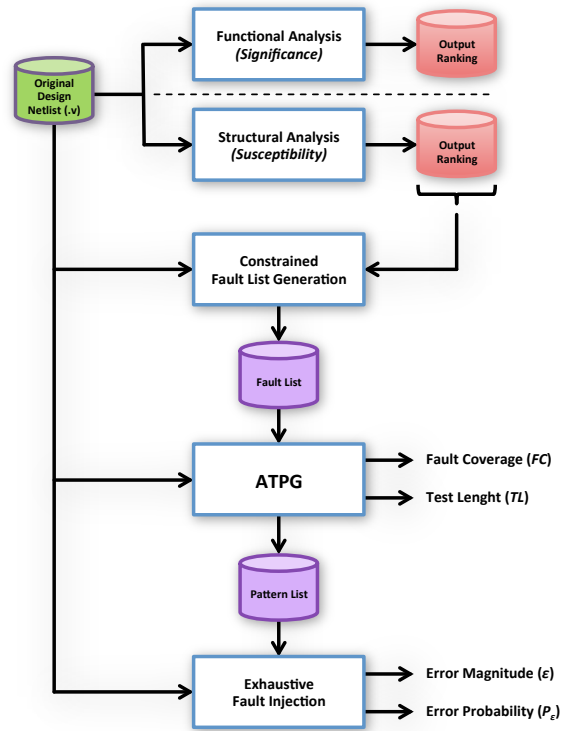


Fig. 1. Approximate Test flow

The Fig. 1 describes the main steps of the proposed AT approach. The starting point is the circuit netlist (original design). The first step is the analysis of the netlist. Actually two types of analysis are performed: the *Functional* and the *Structural*. The goal of each analysis is to rank the circuit outputs based on their *significance* and *susceptibility* respectively. Then, in the second step, a fault list is generated depending on the output ranking previously computed. The fault list contains the fault that must be detected. The third step consists in running an ATPG with the circuit netlist and the generated fault list as inputs. The ATPG provides (in addition to the test patterns) the *Fault Coverage (FC)* and the *Test Length (TL)*. The last step of the proposed flow computes the AC metrics (i.e. P_e and ϵ). Those metrics are computed with the help of an exhaustive fault injection. Next sub-sections provide details on each step of the proposed flow.

A. Functional Analysis – Significance

The functional analysis aims at determining the targeted faults accordingly to the functionality of the circuit. As already discussed, this approach works very well for particular types of circuits. For example, when considering arithmetic ICs, outputs can be easily ranked depending on their weight. The Most Significant Bit (MSiB) of the output data word is the one having the most *significance* while the Low Significant Bit (LSiB) has the low *significance* on the computed result. Consequently, detecting faults affecting the fan-in cone of the MSiB guarantees the functionality with a minimum ϵ . Nevertheless, the *significance* analysis can only be computed on arithmetic ICs and not on random logic ICs. To be more general, the *significance* of the circuit outputs has to be

computed accordingly with the applied workload. The output of this step is the raking of the circuit outputs.

B. Structural Analysis – Susceptibility

Conversely to the functional analysis, we propose to determine the targeted faults by looking only at the circuit structure. In this case, the main idea is to analyze each primary output of the circuit to determine its *susceptibility* as described in [8]. The output *susceptibility* analysis is based on the fact that not all outputs of a circuit have the same *susceptibility*, which is a function of the number of nodes in its fan-in logic cone. It exploits the structural properties of the output fan-in cone to get their relative *susceptibility* estimates. In other words, we aim to identify the outputs more affected by the presence of faults. Compared to the *significance* analysis, the *susceptibility* analysis is only related to the circuit structure. No functional information is required to compute it and thus it can be applied to any digital circuits.

Algorithm 1 shows the pseudo-code of the *susceptibility* analysis methodology. The algorithm starts by reading the pre-place-and-route netlist of the design. Then, it forms groups F_j of all fan-in cells for each circuit output O_j . Once groups are formed the weight W_j of each fan-in cone is calculated by adding together the weights of all cells in the corresponding fan-in cone group. According to the hypothesis that forms the basis of this methodology, cell weight is the number of inputs and outputs of that cell. Ranks are assigned to each output on the basis of their fan-in cone weight using a sort function shown in line 15 of Algorithm 1.

```

1 read(netlist);
2 // Group all fan-in cone cells together for each
  output node
3 foreach  $O_j$  do
4    $F_j \leftarrow O_j.get\_fanin()$ ;
5 end
6 // Get weight of fanin cone of each output
7 foreach  $O_j$  do
8   foreach  $C_i$  do
9     if  $C_i \in F_j$  then
10       $W_j \leftarrow W_j + C_i.get\_pins()$ ;
11    end
12  end
13 end
14 // Sort outputs on the basis of their fanin cone
  weight
15 sort( $O_j, F_j, W_j$ );

```

Algorithm 1. Output susceptibility analysis

The algorithm is further explained by its application to a simple example circuit shown in Fig. 2. The shaded regions mark the boundaries of the two output fan-in cones. The weight parameter (W_i) is given on the top of each gate. The fan-in cones weight (S_j) given on the right of corresponding output is found to be 14 and 10 for O_1 and O_2 respectively.

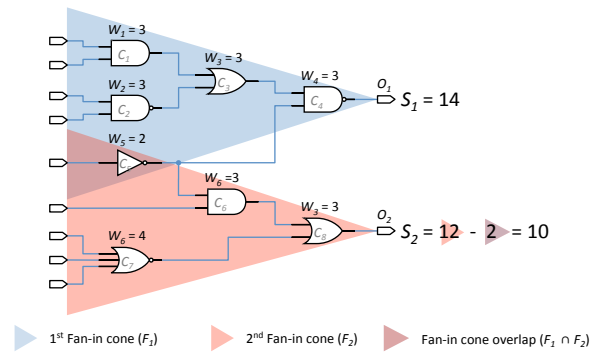


Fig. 2. Application of the susceptibility analysis

According to these figures we can infer that detecting the faults affecting the fan-in cone of O_1 significantly impacts the fault coverage while detecting the remaining faults slightly improves the fault coverage. As for the significance analysis, the output of this step is the raking of the circuit outputs from the Most Susceptible Bit (MSuB) to the Least Susceptible Bit (LSuB). As in the example of Fig. 2 they are O_1 and O_2 respectively.

C. Constrained Fault List Generation

This step takes in input the results of one of the two previous analyses (Functional or Structural). Please note that, if the circuit under consideration is a random logic circuit for which the workload is not known, only the *structural* analysis is possible. Having the circuit outputs ranked, we choose a subset of them that we want to cover during the AT. For each selected output, we add stuck-at-faults affecting its fan-in cone. The resulting fault list is thus composed of all the stuck-at faults observable at the selected outputs. Once again, the generated fault list only contains a sub-set of the total fault list.

Let us come back to the example shown in Fig. 2. For each output we determine the collapsed stuck-at-fault list.

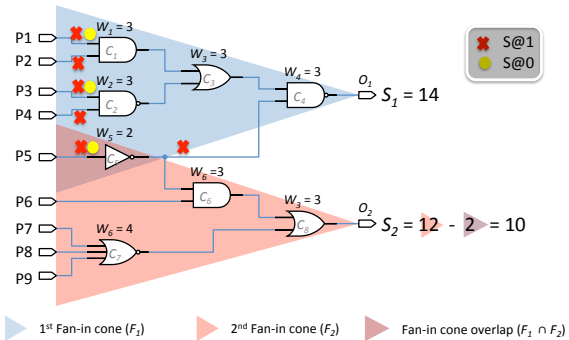


Fig. 3. Fault List generation from O_1

Fig. 3 graphically shows the collapsed fault list obtained by considering the fan-in cone of O_1 . It is composed of 9 stuck-at faults.

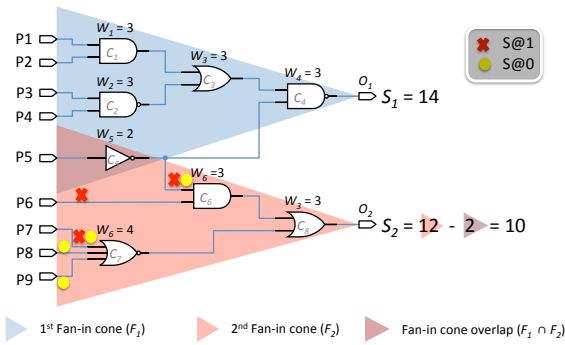


Fig. 4. Fault List generation from O_2

Fig. 4 graphically shows the collapsed fault list obtained by considering the fan-in cone of O_2 . It is composed of 7 stuck-at faults. We detailed this step since it is important to note the fan-out branch at the output of the invert C_5 . In the fault list from O_1 the faults affecting the inverter have been considered. However, In O_2 we must add the faults affecting the branch impacting on the O_2 fan-in cone itself. Otherwise those faults will never be targeted during the Automatic Test Pattern Generation (ATPG).

D. ATPG

For this step of the AT flow, a commercial ATPG tool, with default options (i.e., fault dropping, dynamic compaction and random fill), is used to generate the test patterns. The *Fault Coverage (FC)* and the *Test Length (TL)* are two important outputs since they represent the effectiveness of the proposed AT compared to a deterministic one. We also save the patterns list to further compute the AC metrics.

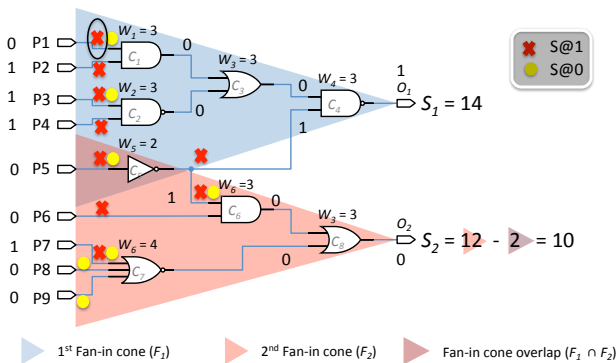


Fig. 5. Test generation example

Fig. 5 gives an example of the test generation. First of all, the target fault is the $S@1$ affecting the primary input P1 (circled red cross in the figure). The ATPG determines the logical values to be applied to the first 5 primary inputs (from P1 to P5) for sensitizing and propagating the fault effect through the circuit gates to reach O_1 . The primary inputs impacting only on the O_2 fan-in cone are simply randomly filled (from P6 to P9). The logical values are shown in the Fig. 5. It is clear that more than one fault is tested for “free” thanks to the input assignments. In this particular example, we test also $S@0$ affecting P3 and P4, $S@1$ affecting P6 and $S@0$

affecting P7. Thus, we can also detect faults affecting O_2 cone thanks to the random filling.

TABLE I. EXAMPLE

Outputs	#. Faults	Detected Faults		FC(%)	TL
		O_1	O_2		
O_1	9	9	5	87.5	6
O_2	7	3	7	100	2

Table I summarizes the test generation for the given example. Let us consider the first row. It reports the ATPG data executed by using the O_1 fault list. The achieved fault coverage is 87.5% and 6 test patterns are generated. The column “detected faults” gives the detail per each fault list. It can be seen that targeting the O_1 fault list will result in detecting for “free” 5 faults of the O_2 fault list thanks to the random fill.

E. Exhaustive Fault Injection

Error Probability (P_e) and the *Error Magnitude (ϵ)* are metrics allowing the evaluation in the AC context. P_e represents the ratio of erroneous responses among the total responses that the circuit can produce. ϵ gives the hamming distance between the erroneous response and the golden one. P_e can be computed for any digital circuits while ϵ computation is only possible for arithmetic circuits or when the workload is known. Since we do not consider a particular workload, we compute both metrics by running an exhaustive fault injection campaign. All possible stuck-at-faults are injected (one after the other) and the exhaustive pattern list is simulated. It is worth to mention that this exhaustive fault injection can only be applied for small circuits (i.e., with a limited number of outputs). In this work is used only for validation, in the general case the workload has to be known in order to compute the metrics. Responses are then compared to the golden ones to compute P_e metric. ϵ metric is obtained in the same way but the hamming distance between the simulated fault injection and the golden response is obtained with the help of the output ranking computed during the functional analysis. Once again this is done only for arithmetic circuits since they do not require a workload to determine the weight of the outputs.

III. EXPERIMENTAL RESULTS

The AT flow has been validated on three case studies. All the circuits have been synthesized with a 45nm technology library 0. Table II reports the main characteristics of each circuit in terms of number of primary inputs (#. PIs), primary outputs (#. POs) and logic gates (#. Gates). All the case studies are purely combinational circuits. The first two are arithmetic circuits able to compute 4-bit sum, subtraction and logic operation. Moreover, the ALU2 can also compute comparisons between the two inputs (i.e., equals-to, greater-than and less-than). The last circuit is the c432 from the ISCAS’85 benchmarks suite [11]. We selected these benchmarks in order to have two cases for which both the functional and structural analyses can be applied, while for the last one, only the structural analysis is applied.

TABLE II. CASE STUDIES

Circuit	#. PIs	#. POs	#. Gates
ALU	11	4	51
ALU2	12	7	127
c432	36	7	160

The goal of the validation is to show that the structural analysis can be used instead of the functional one. Moreover, we want to show that the error obtained by the structural analysis is comparable with the one obtained through the functional one. Thus, the proposed structural approach can be adapted to any kind of circuits without the knowledge of the workload. In the next sub-sections we present the results for each case study.

A. ALU

The Fig. 6 shows the block diagram of the ALU. This circuit can compute the standard arithmetic and boolean operation over the two 4-bit inputs.

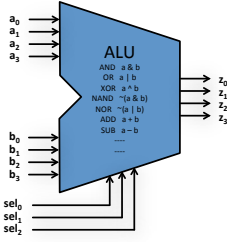


Fig. 6. ALU block diagram

TABLE III. ALU RESULTS

Outputs	FC (%)	TL	P_c	ϵ_{avg}	ϵ_{max}
1	96.58	19	0.199	0.580	7
2	98.16	21	0.140	0.243	3
3	98.95	23	0.060	0.060	1
4	99.47	24	0	0	0

Table III reports the results obtained by applying the AT flow described in the Section II. Since this case study is an arithmetic circuit, we applied both the functional and structural analyses. In this particular case, the order provided by the two analyses is the same: z_3, z_2, z_1 and z_0 . This is because the MSiB (z_3) is the one having the most important fan-in cone and thus it also corresponds to the MSuB. The first column of Table III gives the number of primary outputs considered in the second step for generating the fault list and, consequently the test patterns. Please remember that we selected the primary output for the fault list and the test pattern generation accordingly with their ranking (i.e., from the MSiB/MSuB down to the LSiB/LSuB). Looking at first row, we can see that if we consider only one output (i.e., the fault list contains only the faults affecting the fan-in cone of that primary output), we can achieve a high stuck-at fault coverage (96.58%) with a probability of 0.199 to have a failure at the output (i.e., due to the untested faults). For this case, the average error magnitude (ϵ_{avg}) is 0.580 (i.e., the average deviation from the expected output in the presence of faults) and maximum error magnitude (ϵ_{max}) is 7 (i.e., the highest difference observed between the fault-free circuit and the faulty one). Please consider that a maximum deviation of 7 in the result correspond to a maximum error of 46.67% (i.e., 7 over 15).

Then, by adding the remaining outputs we slightly increase the fault coverage, to reach at the end 99.47%. Considering the test length, we can notice that the number of test patterns varies from 19 up to 24. Thus, only considering one output we can reduce the test length of 12.5%. Depending on the final system quality, the test engineer can therefore select the appropriate set of primary output leading to the desired trade-off between fault coverage, test length, P_c and ϵ_{max} .

Note that the last line in Table III represents the case of a standard deterministic test where all stuck-at-faults are considered during the test generation (i.e., the upper bound of the FC and TL).

B. ALU2

The Fig. 7 shows the block diagram of the ALU2. This circuit can compute the standard arithmetic and boolean operation over the two 4-bit inputs and, in addition, comparison between the inputs.

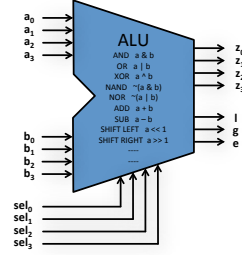


Fig. 7. ALU2 block diagram

Even if this circuit is quite simple in terms of netlist and number of inputs and outputs (as reported in Table II), it is interesting since the functional analysis is not so obvious. Clearly, z_3 is still the MSiB of the data outputs, but what is the weight of the logical outputs (l , g , and e)? Once again, this question can only be answered by the knowledge of the workload. In the following we consider two possible scenarios:

- **S1:** the logical outputs are more significant than the data outputs. Thus the Functional analysis will provide the rank l, g, e, z_3, z_2, z_1 and z_0 .
- **S2:** the logical outputs are less significant than the data outputs. Thus the Functional analysis will provide the rank z_3, z_2, z_1, z_0, l, g and e .

Let us consider S1 as the target scenario for our experiment, so that we can observe what happen when the output ranking is different for *functional* and *structural* analysis.

TABLE IV. FUNCTIONAL ANALYSIS: SCENARIO S1

Outputs	FC (%)	TL	P_c	ϵ_{avg}	ϵ_{max}
1	76.19	27	0.287	1.551	15
2	76.62	29	0.257	1.328	15
3	77.16	31	0.250	1.271	15
4	94.83	55	0.192	0.536	7
5	97.52	62	0.145	0.245	3
6	98.28	66	0.082	0.082	1
7	100.00	75	0	0	0

TABLE V. STRUCTURAL ANALYSIS: SCENARIO S2

Outputs	FC(%)	TL	P_ϵ	ϵ_{avg}	ϵ_{max}
1	85.56	38	0.256	0.592	7
2	88.36	45	0.209	0.302	3
3	89.76	50	0.147	0.141	1
4	93.10	57	0.067	0.063	1
5	99.35	70	0.039	0.038	1
6	99.89	71	0.007	0.007	1
7	100.00	72	0	0	0

Tables IV and 0 report the results obtained by using only the *functional* and the *structural* analysis respectively. We can immediately compare the tables. Let us consider the first row of both the cases. For the *functional* analysis, when considering only one output and its related fan-in cone set of stuck-at faults, we achieve 76% of FC. Conversely, when the output is provided by the *structural* analysis we can reach 85% of FC. There is a quite huge difference of 9% of FC. The P_ϵ and ϵ_{avg} , ϵ_{max} metrics also show a significant difference between the two cases. The ones associated to the *structural* analysis (see Table 0) are lower than those obtained by using the *functional* analysis. Especially the ϵ_{avg} and ϵ_{max} are reduced from 1.55 to 0.59 and from 15 down to 7 respectively. Which correspond to 2.6 times reduction in ϵ_{avg} and a 2.1 times reduction in ϵ_{max} when comparing our *structural* analysis with the *functional* one. Finally, both the analysis lead us a maximum reduction of about 45% of test length compared to the upper bound case (i.e. the last line of the tables where all stuck-at faults are considered during the test generation).

Similar to the first case study (see Section III.A), we can comment that depending on the final system quality, the test engineer can therefore select the appropriate analysis leading to the desired trade-off between fault coverage, test length, P_ϵ and ϵ metrics value.

However, a more interesting comment is about the differences between the *structural* and *functional* analysis. In this case, the structural analysis is even better than the functional one. Due to lack of results, we cannot claim that it is always true, but it is a non-intuitive result: only considering the structural information we can improve the coverage of the functionalities of the target circuit since both P_ϵ and ϵ are reduced.

C. c432

The last case study is the public available benchmark c432 [11]. As reported in Table II it has 36 primary inputs. Even if this number is not so high, it becomes to be not interesting to determine the P_ϵ by using the exhaustive fault injection (the complexity will rapidly increases). Indeed, this case study has been selected in order to put the study in a context close to real cases where the P_ϵ and ϵ can only be computed with a given workload. Thus, the Table VI does not provide any value for P_ϵ and ϵ .

Reported results are indeed interesting: by considering only one output (i.e., the MSuB), we can achieve the 97.32% of FC.

On the other hand, if the complete fault list is generated the FC is 97.88%. So we simply loose 0.5% of fault coverage. Looking at the test complexity, we can reduce it by 4% if we consider only the MSuB output for the fault list generation.

TABLE VI. C432 RESULTS

Outputs	FC(%)	TL
1	97.32%	65
2	97.77%	67
3	97.88%	68
4	97.88%	68
5	97.88%	68
6	97.88%	68
7	97.88%	68

IV. CONCLUSION

The contribution of this work was to investigate an approach opposite w.r.t. the state-of-the-art. Instead to classify the faults according to the applied workload, we exploit a structural analysis to determine the most vulnerable circuit elements and thus generate test patterns for these elements. Preliminary results indicate that the resulting method, called *Approximate Test* can really lead to provide benefits in terms of test time reduction. Test engineer can select the desired trade-off between quality and test complexity. The proposed *structural* analysis methodology seems interesting and promising step toward making the AT applicable for any kind of circuits.

REFERENCES

- [1] G. Gielen, et al., "Emerging yield and reliability challenges in nanometer CMOS technologies", in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1322-1327, 2008
- [2] Sparsh Mittal, "A Survey of Techniques for Approximate Computing", *ACM Computing Surveys (CSUR)*, Volume 48, Issue 4, May 2016.
- [3] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," in *IEEE Design & Test*, vol. 33, no. 1, pp. 8-22, Feb. 2016.
- [4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," *2013 18th IEEE European Test Symposium (ETS)*, Avignon, 2013, pp. 1-6.
- [5] V. Chippa et al., "Analysis and characterization of inherent application resilience for approximate computing", in *Proc. 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13)*, May 2013, pp. 1-9.
- [6] K.-J. Lee, T.-Y. Hsieh, and M. A. Breuer, "Efficient Overdetection Elimination of Acceptable Faults for Yield Improvement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 5, pp. 754-764, May 2012.
- [7] Suraj Sindia, Vishwani D. Agrawal, "Tailoring Tests for Functional Binning of Integrated Circuits", *IEEE 21st Asian Test Symposium*, pp. 95-100, 2012
- [8] I. Wali, B. Deveautour, A. Virazel, A. Bosio, P. Girard, M. Sonza Reorda, "A Low-cost Susceptibility Analysis Methodology to Selectively Harden Logic Circuits", *IEEE 21st European Test Symposium*, 2016
- [9] A. Momeni, J. Han, P. Montuschi and F. Lombardi, "Design and Analysis of Approximate Compressors for Multiplication," in *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984-994, April 2015.
- [10] Nangate Inc., "45nm open cell library", <http://www.nangate.com>.
- [11] <https://filebox.ece.vt.edu/~mhsiao/iscas85.html>